ENSIAME

Optimisation:

De l'estimation paramétrique à l'apprentissage, une ballade entre théorie et pratique

S. Delprat

Chapitre 5 – Optimisation sans information sur les dérivées du critère et des contraintes

1) Méthode du simplexe
2) Algorithmes génétiques

Dérivation numérique

La dérivée du critère peut ne pas être connue:
$\Rightarrow$ Parce que le critère n'est pas continu
$\Rightarrow$ Parce qu'on optimise des variables entière et/ou binaires

Ou le calcul de la dérivée peut être trop couteux.

=> On s'intéresse aux méthodes dites « dérivative-free », s'est à dire qui ne requièrent aucune dérivée.

Méthode du simplexe

Référence : Nelder-Mead 1965

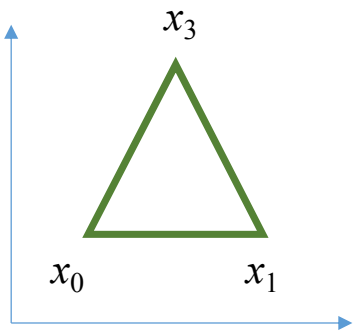
Un simplexe est une généralisation d'un triangle dans un espace à n dimensions.

Idée: En partant d'un simplexe initial, on l'étire (ou le rétrécie) et le déplace jusqu'à convergence.

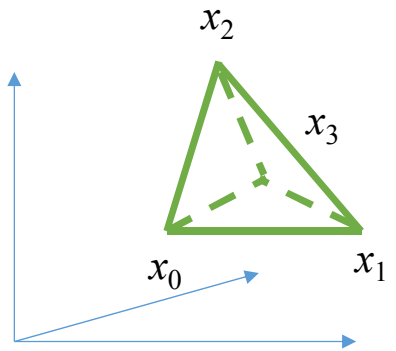
Le problème à résoudre: $f\left(x^*\right) = \min_x f\left(x\right)$ $\quad f : \mathbb{R}^n \to \mathbb{R}$

La fonction f n'est pas nécessairement dérivable.

n=2 : Le simplexe est un triangle
Simplexe à n+1=3 points



n=3 : Le simplexe est un prisme
Simplexe à n+1=4 points



Méthode du simplexe

Paramètres:

x_0 : solution initiale

$\alpha \geq 1$: Coefficient de réflexion

$\gamma \geq 1$: Coefficient d'expansion

$\sigma < 1$: Coefficient de réduction

$\beta < 1$: Coefficient de contraction

ε : Valeur pour tester la convergence

m : Taille du simplexe

Etape n°1: Génération du simplexe initial

$P(:,1) = x_0$
POUR i=1:m-1
  $P(:,i+1) = x_0$
  $P(i,i+1) = P(i,\ i+1)$ x 1,05
  SI $P(i,i+1)$==0 ALORS
    P(i,i+1)=10^{-8}
  Fin si
Fin pour

POUR i=1:m
  $F(i) = f(P(:,i))$
Fin pour

P: matrice contenant les m sommets du simplexe

F: Tableau contenant les m valeurs de la fonction pour chaque sommet

Méthode du simplexe

Tant que critère non satisfait

Step 2: On trie les sommets du simplexe dans l'ordre croissant des valeurs de la fonction

$P(:,1)$ Meilleur point $P(:,m)$ Plus mauvais point

Step 3: Phase de réflexion

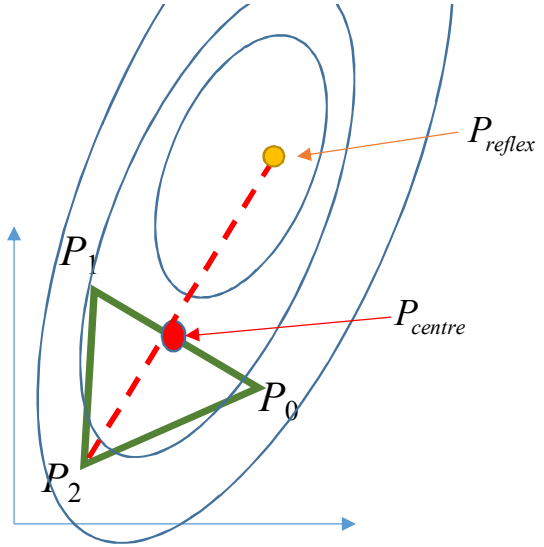
Calcul du « centre » P_{centre} des m-1 premier sommets

$$P_{centre} = \frac{1}{m-1}\sum_{i=1}^{m-1} P(:,i) \qquad F_{centre} = f\left(P_{centre}\right)$$

On essaye de s'éloigner du plus mauvais point $P(:,m)$

$$P_{reflex} = (1+\alpha)P_{center} + \alpha P(:,m) \qquad F_{reflex} = f\left(P_{reflex}\right)$$

α *Permet de contrôler la taille du pas*



P_0 : point le plus mauvais

Méthode du simplexe

On évalue la situation

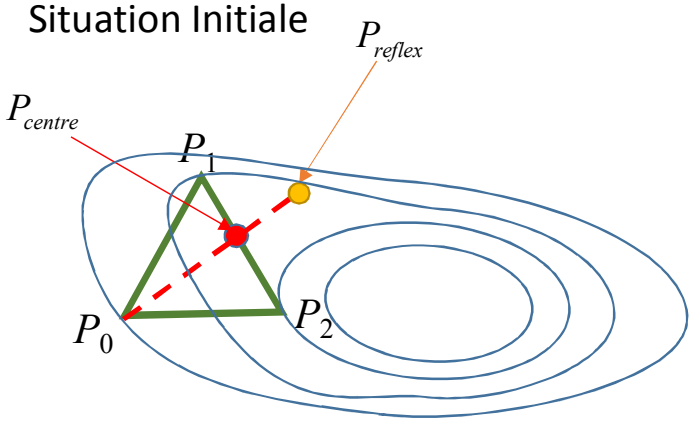
Si ($F(1) \leq F_{reflex}$) & ($F_{reflex} < F(m\text{-}1)$ ALORS

Le point n'est ni meilleur ni pire

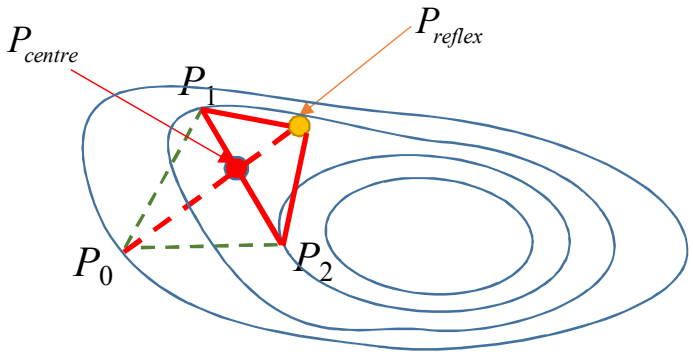
$$P(:,m) = P_{reflex} \qquad F(m) = F_{reflex}$$

Goto Step 2

Situation Initiale



Itération suivante



Méthode du simplexe

On évalue la situation

Si ($F(1) \leq F_{reflex}$) & ($F_{reflex} < F(m\text{-}1)$ ALORS

Le point n'est ni meilleur ni pire

$$P(:,m) = P_{reflex} \qquad F(m) = F_{reflex}$$

Sinon

Si ($F_{reflex} \leq F(1)$)

La réflexion améliore la situation

Phase d'expansion => On pousse plus loin

$$P_{\exp} = (1+\gamma) P_{center} + \gamma P_{refelex} \qquad F_{\exp} = f(P_{\exp})$$

Si ($F_{exp} \leq F_{reflex}$) ALORS

$$P(:,m) = P_{\exp} \qquad F(m) = F_{\exp}$$

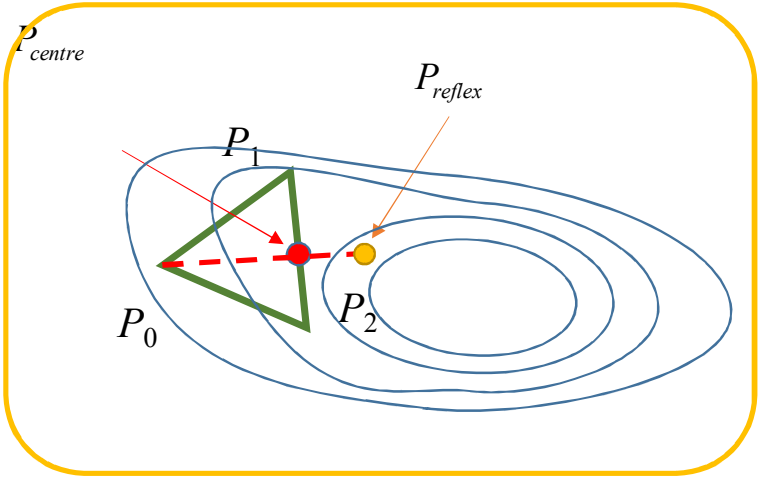
SINON

$$P(:,m) = P_{reflex} \qquad F(m) = F_{reflex}$$

FSI

Goto Step 2

On garde la meilleure des solution



Méthode du simplexe

SINON

Donc on a ($F_{reflex} \geq F(m\text{-}1)$)

Si ($F_{reflex} > F(1)$)

Phase de Contraction : Calcul de 2 simplexes plus petits

$$P^{out}_{contract} = (1+\beta) P_{center} + \beta P_{refelex} \qquad F^{out}_{contract} = f\left(P^{out}_{contract}\right)$$

$$P^{in}_{contract} = (1+\beta) P_{center} + \beta P(:,m) \qquad F^{in}_{contract} = f\left(P^{in}_{contract}\right)$$

Si $\left(F^{out}_{contract} < F_{reflex}\right) \& \left(F_{reflex} < F(m)\right)$ ALORS

$$P(:,m) = P^{out}_{contract} \qquad F(m) = F^{out}_{contract}$$

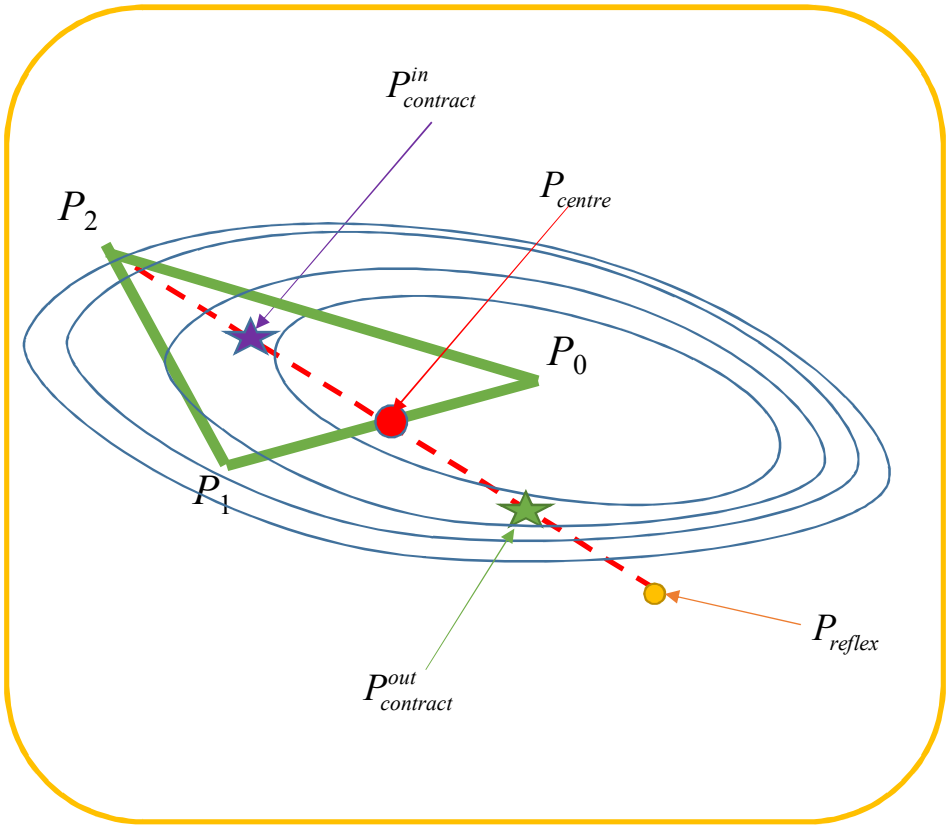
SINON

Si $\left(F^{in}_{contract} < F(m)\right) \& \left(F(m) \leq F_{reflex}\right)$ ALORS

$$P(:,m) = P^{in}_{contract} \qquad F(m) = F^{in}_{contract}$$

SINON

Phase de Retrecicement



Méthode du simplexe

Phase de Retrecicement

On rapetisse le simplexe vers le meilleur point

Pour i=2:m

$$P(:,i) = (1-\sigma)P(:,1) + \sigma P(:,1)$$

$$F(i) = f\left(P(:,i)\right)$$

Fin pour

Fin si

Fin si

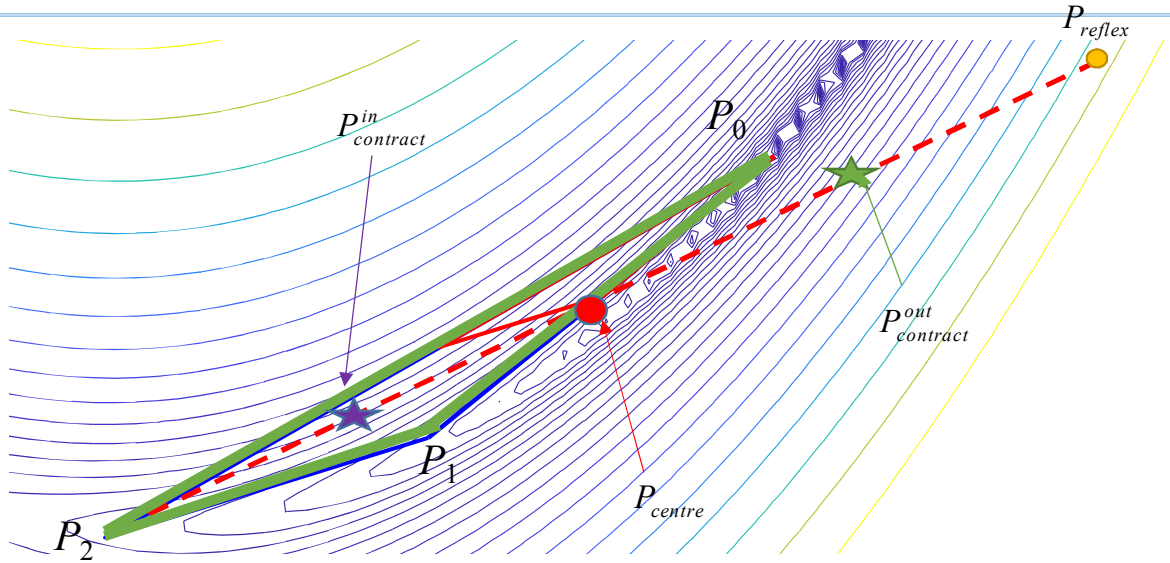
Fin si

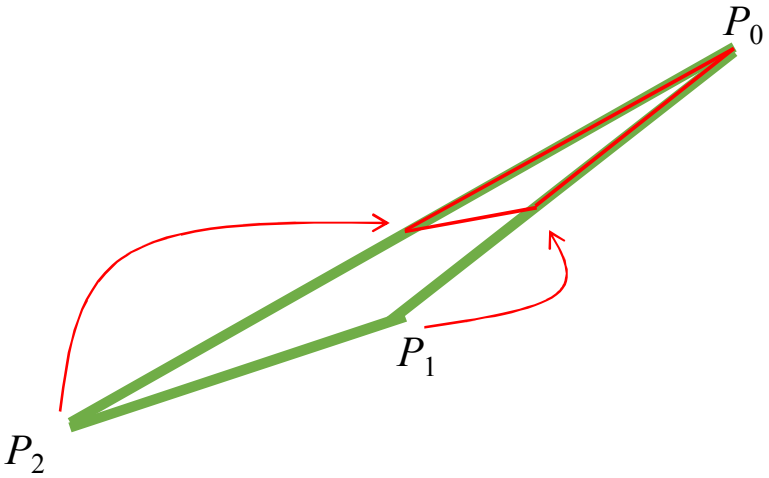
Fin si

$$TailleSimplexe = \sqrt{\frac{1}{m-1}\sum_{i=1}^{m}\left(F(i) - F_{centre}\right)^2}$$

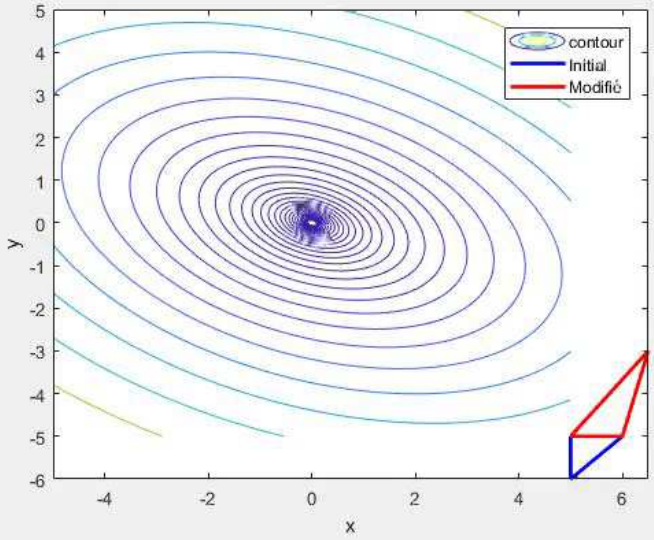
$$Critere = TailleSimplexe < SeuilTaille$$

Fin Tant Que



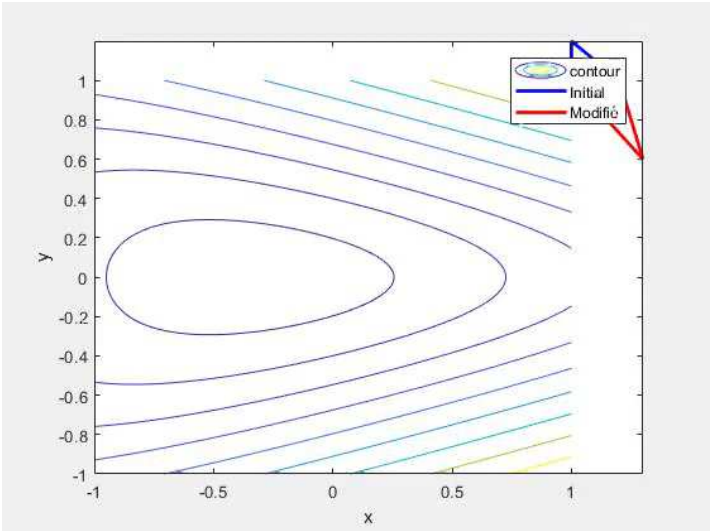


Méthode du simplexe

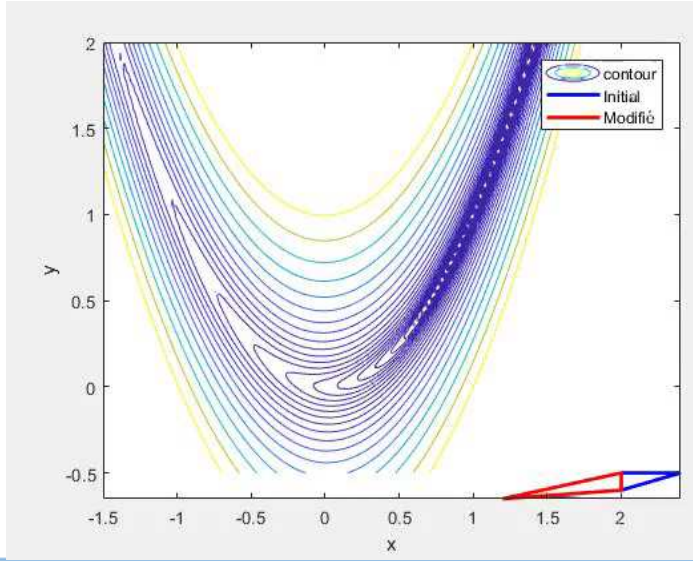


Fonction quadratique

Fonction test



Rosembrock



Méthode du simplexe

```
clear all;
close all;
clc;
global nFunction f2
nFunction=3;

NoFonction=1;
ChoixFonction;

%x0=[-2;-2];

% Simplex paramters
alpha=1; % Reflexion paramters
gamma=2; % Expenson coefficient
Sigma=0.5; % Shrink coefficient
beta=0.5;  % Contraction coefficnet
EpsF=1e-5; % Convergence on the f value

% Initial simplex
m=size(x0,1)+1;
Simplex.P=zeros(m-1,m);
Simplex.P(:,1)=x0;
for i=1:m-1
    Simplex.P(:,i+1)=x0;
    if Simplex.P(i,i+1)==0
        Simplex.P(i,i+1)=1e-8;
    end
    Simplex.P(i,i+1)=Simplex.P(i,i+1)*1.2;
end
```

```
for i=1:m
    Simplex.F(i)=f(Simplex.P(:,i));
end
```

Méthode du simplexe

```
ended=0;
while ended==0
    % Reorder the simplex in increasing order
    [Simplex.F,iOrder]=sort(Simplex.F);
    Simplex.P=Simplex.P(:,iOrder);

    oldSimplex=Simplex;

    % Compute center of the m-1 lower values
    Simplex.Pcenter=mean(Simplex.P(:,1:m-1),2);
    Simplex.Fcenter=f(Simplex.Pcenter);

    % Reflextion
    fprintf('Reflexion\n');
    Preflex=(1+alpha)*Simplex.Pcenter-alpha*Simplex.P(:,m);
    Freflex=f(Preflex);
    Simplex.Preflex=Preflex;
    Simplex.Freflex=Freflex;

    if (Simplex.F(1)<=Freflex)&(Freflex<Simplex.F(m-1))

        % Reflexion génère un point "intermédiaire" ni meiller ni pire
        Simplex.P(:,m)=Preflex;
        Simplex.F(m)=Freflex;
        Phase=1;
    else
```

Méthode du simplexe

```
% La réflexion génère soit un meilleur point soit un point plus
% mauvais
if Freflex<Simplex.F(1)
    % Meilleure solution trouvée
    % On tente l'expension pour "pousser plus loin"
    fprintf('Expansion : ');
    Pexpand=(1-gamma)*Simplex.Pcenter+gamma*Preflex;
    Fexpand=f(Pexpand);
    Simplex.Pexpand=Pexpand;
    Simplex.Fexpand=Fexpand;

    Phase=2;
    if Fexpand<Freflex
        fprintf(' réussie\n');
        % Expension réussie, on garde ce point
        Simplex.P(:,m)=Pexpand;
        Simplex.F(m)=Fexpand;
    else
        % Expension échour: on garde le point de réflexion
        fprintf(' échoue\n');
        Simplex.P(:,m)=Preflex;
        Simplex.F(m)=Freflex;
    end
else
```

Méthode du simplexe

```
            % Reflexion échoue
            fprintf('Reflexion échoue\n');
            % Le point de réflexion est pire
            fprintf('Contraction\n');
            Pcontract_out=beta*Preflex+(1-beta)*Simplex.Pcenter;
            Fcontract_out=f(Pcontract_out);
            Pcontract_in=beta*Simplex.P(:,m)+(1-beta)*Simplex.Pcenter;
            Fcontract_in=f(Pcontract_in);
            Simplex.Pcontract_out=Pcontract_out;
            Simplex.Fcontract_out=Fcontract_out;
            Simplex.Pcontract_in=Pcontract_in;
            Simplex.Fcontract_in=Fcontract_in;
            if (Freflex < Simplex.F(m)) && (Fcontract_out<= Freflex)
                Simplex.P(:,m) = Pcontract_out;
                Simplex.F(m)   = Fcontract_out;
                Phase=3;
            else
                if(Simplex.F(m) <= Freflex) && (Fcontract_in < Simplex.F(m))
                    %Contract Inside
                    Simplex.P(:,m)=Pcontract_in;
                    Simplex.F(m) = Fcontract_in;
                    Phase=4;
                else
                    % Le point issu de la contraction est pire que la celui de la réflexion et l'original
                    Phase=5;
                    fprintf('Rétrecissement\n');
                    for i=2:m
                        Simplex.P(:,i)=(1-Sigma)*Simplex.P(:,1)+Sigma*Simplex.P(:,i);
                        Simplex.F(i)=f(Simplex.P(:,i));
                    end
                end
            end

        end
    end
```

Méthode du simplexe

```
    display(f2,xmin,xmax,ymin,ymax,oldSimplex,Simplex,Phase) ;
    drawnow;
    if Phase>4
        disp('contract');
    end
    %pause(0.5);
    ended=sqrt(sum(Simplex.F-Simplex.Fcenter).^2/(m-1))<EpsF;
end
```

```
function y=f(x)
global nFunction f2
nFunction=nFunction+1;
y=f2(x);
end

function DisplaySimplexe(S,coul)
%plot3(S.P(1,[1 2 3 1]),S.P(2,[1 2 3 1]),S.F([1 2 3 1]),coul,'linewidth',2);
plot(S.P(1,[1 2 3 1]),S.P(2,[1 2 3 1]),coul,'linewidth',2);
end
```

```
function display(f2,xmin,xmax,ymin,ymax,oldSimplex,newSimplex,Phase)

% Affichage de la fonction et des deux simplexes
X=linspace(xmin,xmax,121);
Y=linspace(ymin,ymax,122);
Z=zeros(length(Y),length(X));
for i=1:length(X)
    for j=1:length(Y)
        Z(j,i)=f2([X(i);Y(j)]);
    end
end

figure(10);
clf;
%surf(X,Y,Z,'EdgeColor','none');
levels=logspace(-2,2,30);
contour(X,Y,Z,levels);
hold on;
xlabel('x');
ylabel('y');
hold on;

DisplaySimplexe(oldSimplex,'b');
DisplaySimplexe(newSimplex,'r');
legend('contour','Initial','Modifié');

if (Phase>2)
    % On teste 2 contractions

plot3(newSimplex.Pcontract_in(1),newSimplex.Pcontract_in(2),newSimplex.Fc
act_in,'r*');

plot3(newSimplex.Pcontract_out(1),newSimplex.Pcontract_out(2),newSimplex.
tract_out,'b*');

plot3(newSimplex.Preflex(1),newSimplex.Preflex(2),newSimplex.Freflex,'go'
end
end
```

Méthode du simplexe

```
function display(f2,xmin,xmax,ymin,ymax,oldSimplex,newSimplex,Phase)

% Affichage de la fonction et des deux simplexes
X=linspace(xmin,xmax,121);
Y=linspace(ymin,ymax,122);
Z=zeros(length(Y),length(X));
for i=1:length(X)
    for j=1:length(Y)
        Z(j,i)=f2([X(i);Y(j)]);
    end
end

figure(10);
clf;
levels=logspace(-2,2,30);
contour(X,Y,Z,levels);
hold on;
xlabel('x');
ylabel('y');
hold on;

DisplaySimplexe(oldSimplex,'b');
DisplaySimplexe(newSimplex,'r');
legend('contour','Initial','Modifié');

if (Phase>2)
    % On teste 2 contractions

plot3(newSimplex.Pcontract_in(1),newSimplex.Pcontract_in(2),newSimplex.Fcontr
act_in,'r*');

plot3(newSimplex.Pcontract_out(1),newSimplex.Pcontract_out(2),newSimplex.Fcon
tract_out,'b*');

plot3(newSimplex.Preflex(1),newSimplex.Preflex(2),newSimplex.Freflex,'go');
end
end
```

1) Méthode du simplexe
2) Algorithmes génétiques

Algorithme génétique

Les algorithmes génétiques sont une méthode d'optimisation évolutionnaire bio-inspirée.

Permet :

- d'optimiser des valeurs réelles, entières et/ou binaire
- d'éviter potentiellement des minimum locaux (éventuellement de converger dans vers un minimum global)

Mais

- les performances dépendent beaucoup des réglages
- En général, aucune garantie d'obtenir un minimum global

Avantages:

- Ne nécessite pas d'information sur la dérivée
- Fournit un ensemble de « bonnes » solutions
- Bcp d'opérations parallélisables

/https://www.tutorialspoint.com/genetic_algorithms/

Inconvénients:

- Aucune garantie de convergence
- Potentiellement bcp d'évaluation du critère
- Gestion des contraintes complexe

Algorithme génétique

Initialement, les algorithmes génétiques ont été développés pour des problèmes où les solutions sont des nombres binaires où entiers.

La version présentée ici est adaptée aux nombres réels.

Le codage d'une solution du problème de minimisation doit être travaillée.

Algorithme génétique - *vocabulaire*

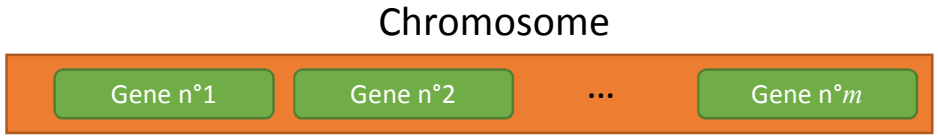
$$\min_x f(x)$$

Population :
Ensemble de n valeurs possibles

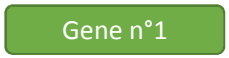
$$P = \{x_i\} \quad i = 1..n$$

Chromosome :
Une valeur x_i contenant m paramètres à optimiser

$$x_i = \left(x_i^j\right) \quad j = 1..m$$



Gène :
C'est un des m paramètres à optimiser



Algorithme génétique – *algorithme*

Etape n°1: Génération d'une population initiale $P = \{x_i\} \quad i = 1..n$

La population doit avoir une « grande » diversité.

Il faut maximiser *une distance* entre les individus

Si dispose d'informations, on peut forcer la présence de quelques individus « performants »

Le reste des individus est tiré aléatoirement

Algorithme génétique – *algorithme*

```
clear all;
close all;
clc;

f=@(x) (x(1,:).^2+x(2,:).^2).*(10+sin(5*x(1,:))+cos(5*x(2,:)));

nindividus=100;
nx=10;
Range=2;
Population=GenerePopulationInitiale(f,nx,nindividus,Range);

function DisplayPopulation(f,Population)
Zpop=f(Population);
[X,Y]=meshgrid(linspace(-2,2,40),linspace(-2,2,41));
Z=X;
Z(:)=f([X(:)';Y(:)']);
figure(15);
clf;
surf(X,Y,Z);
hold on;
plot3(Population(1,:),Population(2,:),Zpop,'r*');
grid on
drawnow;
end
```

```
function d=Diversite(Population)
d=0;
for i=1:length(Population)
    for j=1:i-1
        d=d+norm(Population(:,i)-Population(:,j));
    end
end
end
function
Population=GenerePopulationInitiale(f,nx,nindividus,Range)
NbEssais=100;
BestDiversity=-inf;
for No=1:NbEssais
    NewPopulation=(rand(nx,nindividus)-0.5)*Range*2;
    d=Diversite(NewPopulation);
    if d>BestDiversity
        Population=NewPopulation;
        DisplayPopulation(f,Population)
    end
end
end
```

Algorithme génétique – *algorithme*

Etape n°2: Evaluation de la performance au sein de la population

Les algorithmes génétiques vont faire évoluer certains individus « performants » de la population ?

On cherche à attribuer une probabilité de sélection plus importante aux individus les plus performants:

Exemple de résultat attendu pour une population de 4 individus

Question : comment définir la performance ?

Problème : $\min f(x)$ On ne sait rien de la fonction à minimiser.

$\alpha_i = \frac{1}{f(x_i) - \underline{f}}$: Ne fonctionne que si on connait une borne telle que $\underline{f} > f(x_i)$

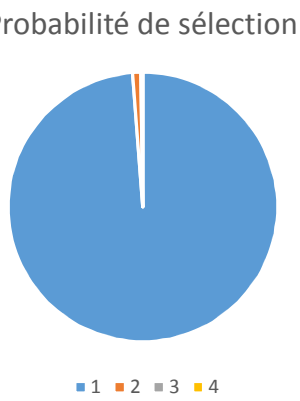
$p_i = \frac{\alpha_i}{\sum_i \alpha_i}$: Normalisation

Algorithme génétique – *algorithme*

Etape n°2: Evaluation de la performance au sein de la population

Individu	1	2	3	4
Fitness	0,001	0,1	0,5	10
α_i	1000	10	2	0,1
p_i	0,98	0,0099	0.0020	9.8804e-05

Seul l'individu n°1 sera sélectionné en pratique. Un seul individu étant quais-sûr d'être sélectionné, la population va s'hyper-spécialiser. L'algorithme perds ainsi toute capacité exploratoire et ne sortira pas d'un minimum local.



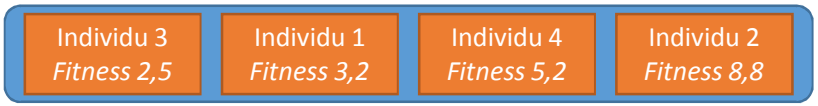
Algorithme génétique – *algorithme*

Etape n°2: Evaluation de la performance au sein de la population

Dans le cas d'un algorithme génétique, on s'intéresse à la variation de performance entre les individu. On peut donc utiliser le rang des individu au lieu de leur performance.

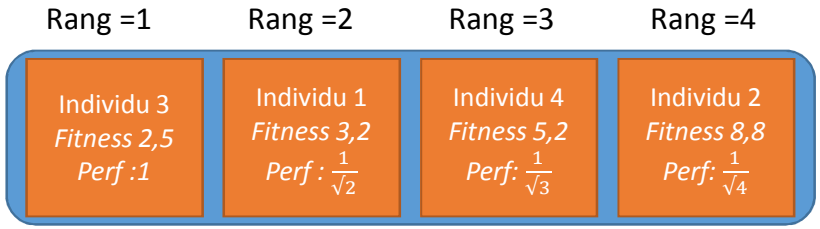
Population:

Etape n°1: On trie les individus selon leur performance



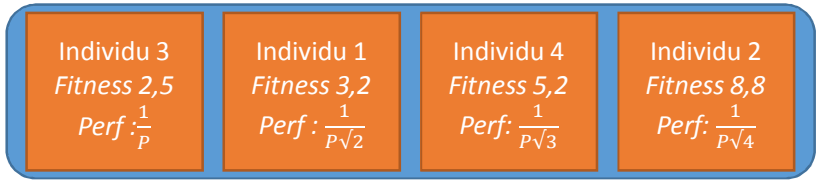
Etape n°2: On calcule la performance en fonction du rang i

$$p_i = \frac{1}{\sqrt{i}}$$



Etape n°3: On normalise la performance

$$p_i' = \frac{p_i}{P} \qquad P = \sum p_i$$

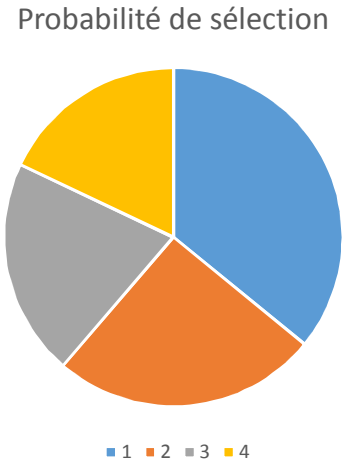


Algorithme génétique – *algorithme*

Etape n°2: Evaluation de la performance au sein de la population

```
function Perf=EvaluationPerf(Fitness)
[~,ordre]=sort(Fitness);
Perf=1./sqrt(ordre);
P=sum(Perf);
Perf=Perf/P;
end
```

Fitness	0,001	0,1	0,5	10
Rang	1	2	3	4
P_i	1	0,71	0,58	0,5
P_i'	0,36	0,25	0,21	0,18



Avantage : Les individus on tous une chance d'être sélectionnés

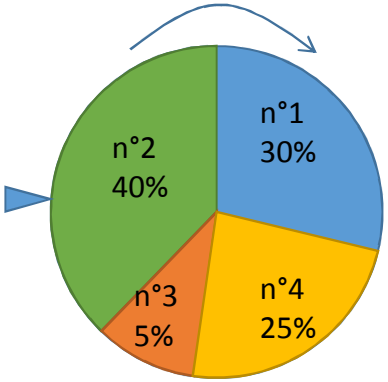
Inconvénient : la performance n'est pas directement liée au Fitness

Algorithme génétique – *algorithme*

Etape n°3: Sélection des parents

On va sélectionner $n_{parents}$ parents autorisés à se reproduire. La probabilité de sélectionner un parent doit être égale à sa performance.

Méthode de la roue de la fortune:



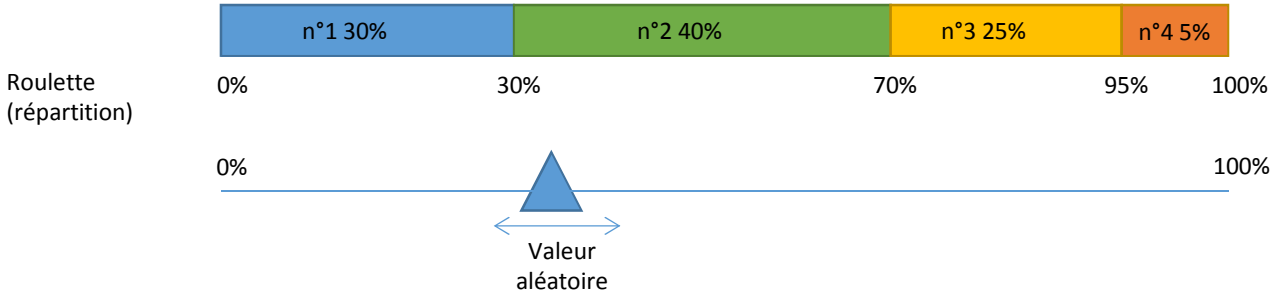
La roue à une probabilité de sélectionner un individu proportionnelle à leur performance

Un individu peut donc être sélectionné plusieurs fois.

Algorithme génétique – *algorithme*

Etape n°3: Sélection des parents

On va sélectionner $n_{parents}$ parents autorisés à se reproduire. La probabilité de sélectionner un parent doit être égale à sa performance.




```
function [Parents,noParent]=SelectionParents(Perf,Population,nParents)
% Selection de nParents dans la population avec une probabilité
% égale à leur performance
Roulette=cumsum(Perf);
Parents=zeros(size(Population,1),nParents);
noParent=zeros(1,nParents);
for i=1:nParents
    r=rand;
    j=find(Roulette>=r,1,'first');
    noParent(i)=j;
    Parents(:,i)=Population(:,j);
end
end
```

Algorithme génétique – *algorithme*

Etape n°3: Sélection des parents

Validation de la procédure: on vérifie la statistique vs la probabilité souhaitée

```
clear all;
close all;
clc;

f=@(x) (x(1,:).^2+x(2,:).^2).*(10+sin(5*x(1,:))+cos(5*x(2,:)));
nindividus=150; nx=2; Range=2;
Population=GenerePopulationInitiale(f,nx,nindividus,Range);
Fitness=f(Population);
Perf=EvaluationPerf(Fitness);

% On vérifie que la probabilité de chaque parents soit bien respectée
m=1000; nparents=50;
noParents=zeros(nparents,m);
for i=1:m
    [~,noParents(:,i)]=SelectionParents(Perf,Population,nparents);
end

% Vérification proba a postériori
nn=numel(noParents);
for i=1:length(Perf)
    % Vérification des probabilitées obtenues
    proba=sum(noParents(:)==i)/nn;
    fprintf('Expected : : %.2f real : %.2f\n',Perf(i)*100,proba*100);
    figure(10);
    hold on;
    plot(i,Perf(i)*100,'r*',i,proba*100,'bo');
end
xlabel('Individu'); legend('Probabilité','Statistique'); ylabel('%')
```

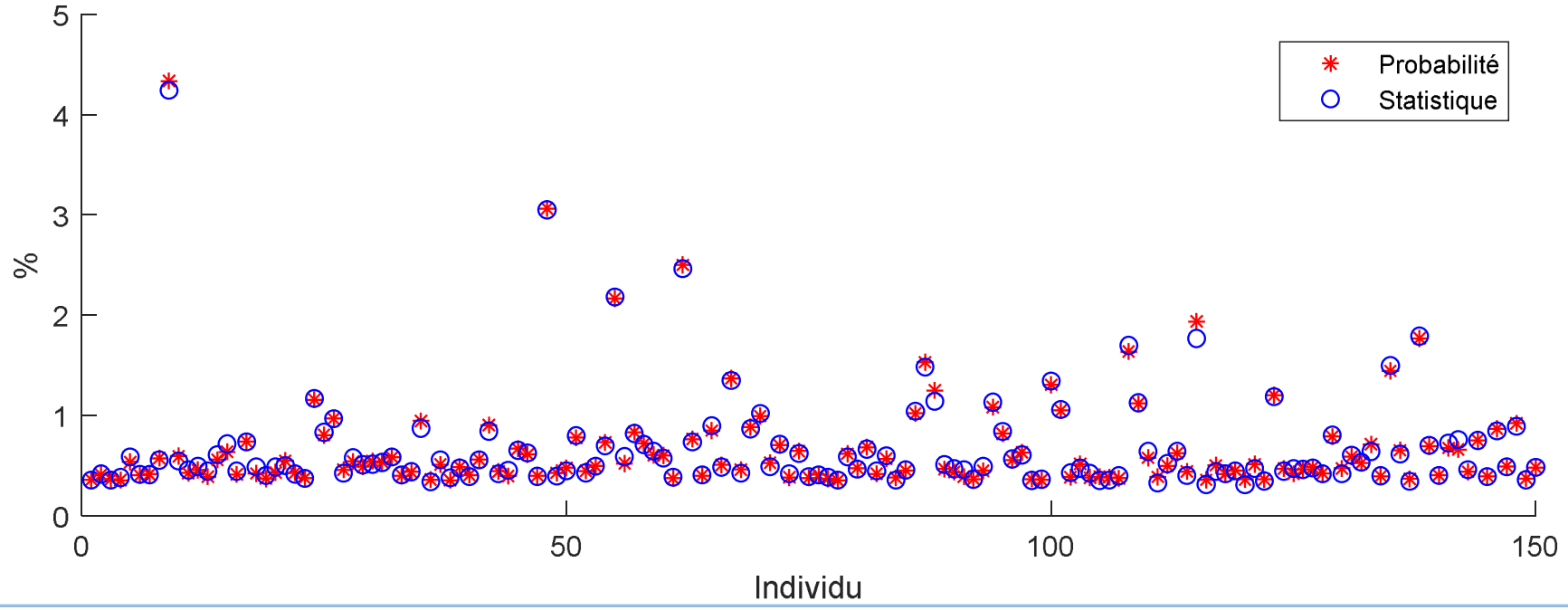
Algorithme génétique – *algorithme*

Etape n°3: Sélection des parents

Validation de la procédure: on vérifie la statistique vs la probabilité souhaitée:

- Population initiale : 150 individus générés aléatoirement
- Nparents=50

Statistiquement, la statistique obtenue est proche de la probabilité espérée
=> Algorithme validé



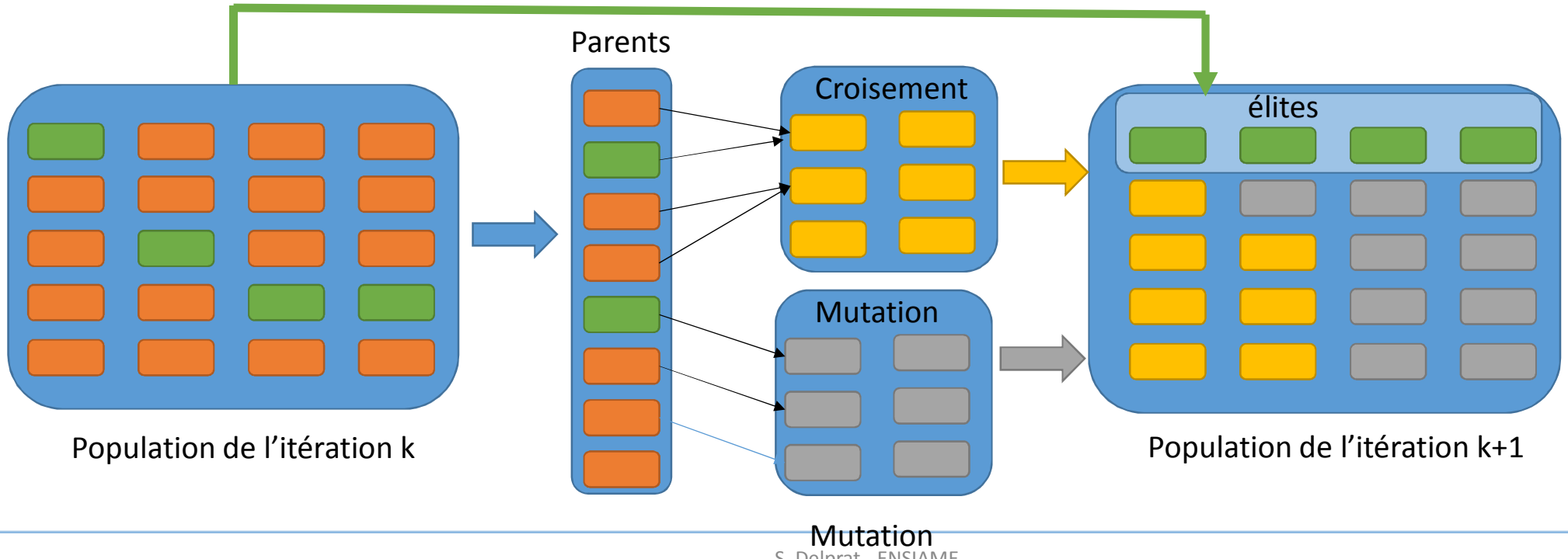
Algorithme génétique – *algorithme*

Etape n°4: Génération d'une nouvelle population

4.1) on sélectionne les n_{elite} individus les plus performants

=> Objectif ne pas perdre les meilleurs individus s'ils n'étaient pas sélectionnés comme parents

4.2) on génère $n - n_{elite}$ à partir des $n_{parents}$ sélectionnés en appliquant des opérateurs génétiques

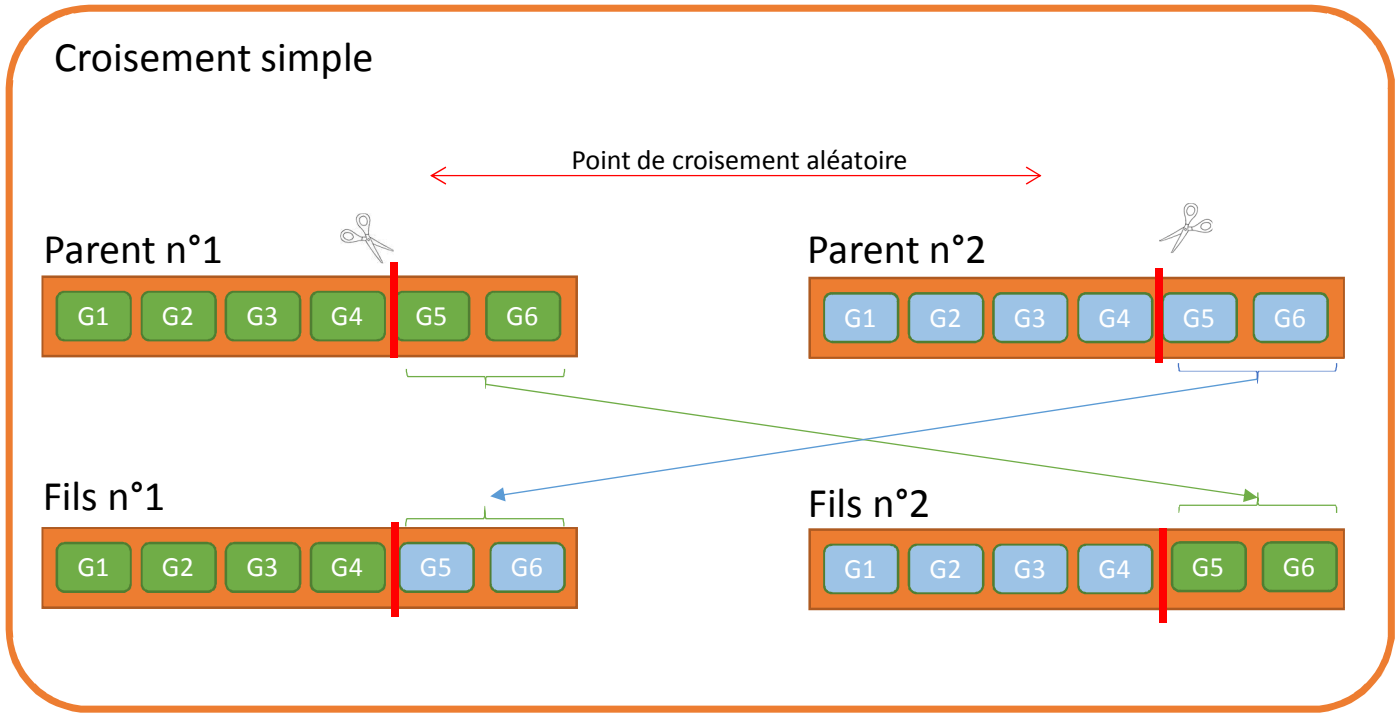


Algorithme génétique – *algorithme*

Etape n°4.2 : Croisements

Afin d'explorer de nouvelles zones, il faut générer de nouveaux individus.

A partir de 2 parents, on va générer 2 nouveaux individus en échangeant de l'information entre les parents

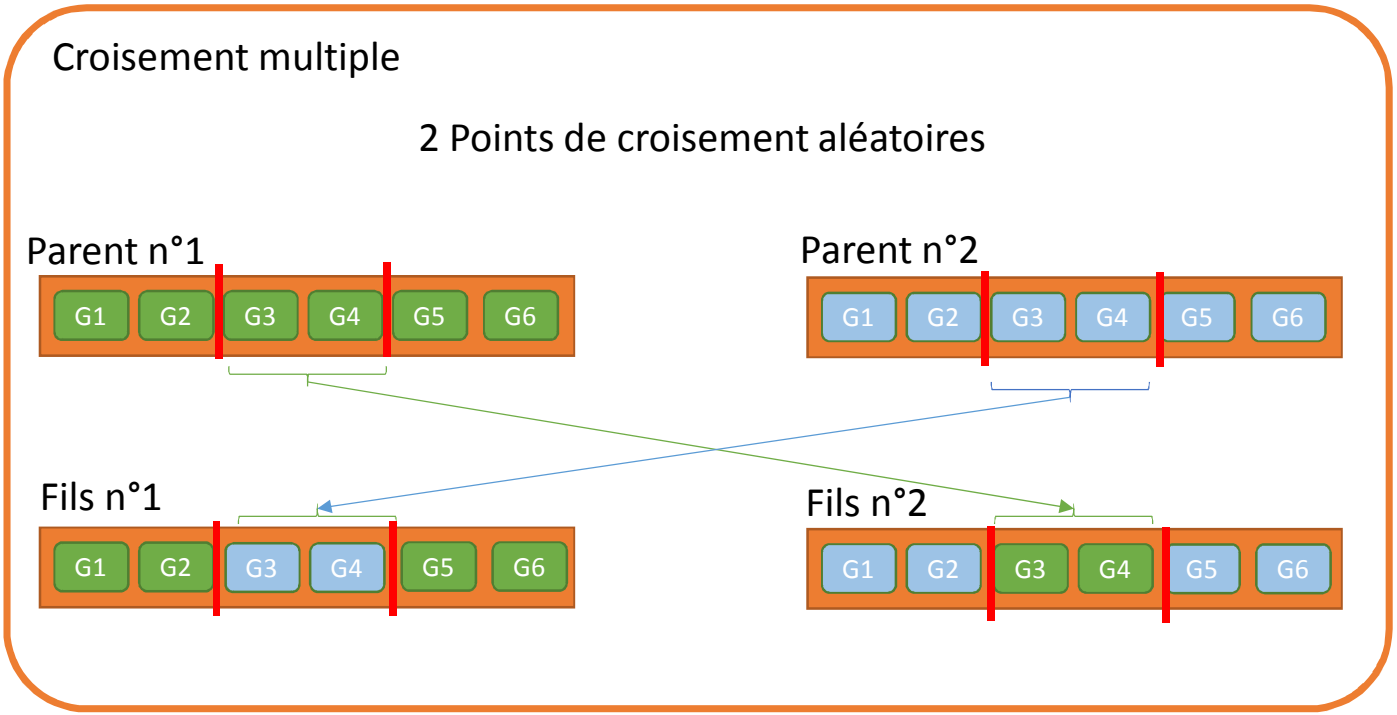


Algorithme génétique – *algorithme*

Etape n°4.2 : Croisements

Afin d'explorer de nouvelles zones, il faut générer de nouveaux individus.

A partir de 2 parents, on va générer 2 nouveaux individus en échangeant de l'information entre les parents



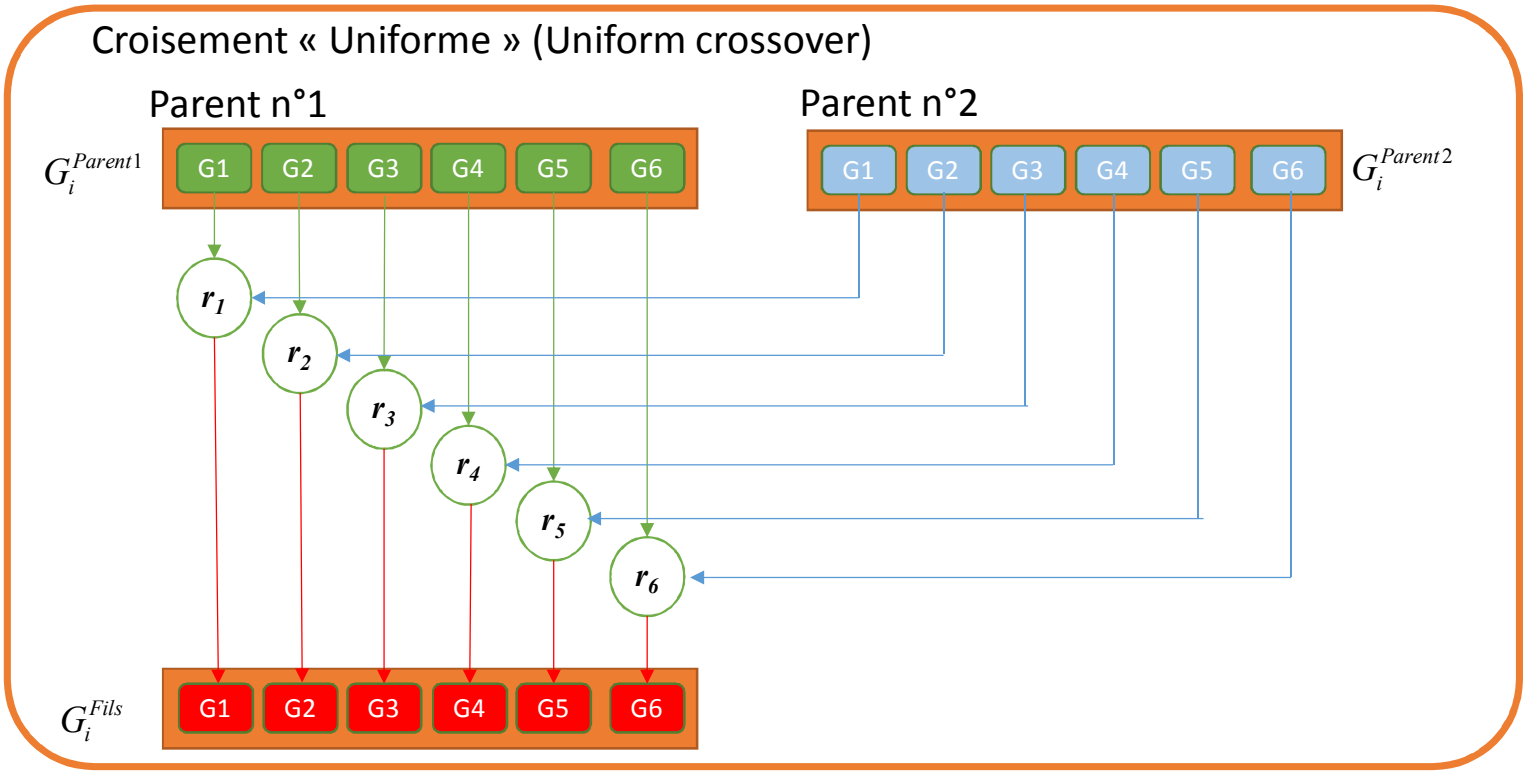
Algorithme génétique – *algorithme*

Etape n°4.2 : Croisement Uniforme

Afin d'explorer de nouvelles zones, il faut générer de nouveaux individus.
A partir de 2 parents, on va générer 2 nouveaux individus en échangeant de l'information entre les parents

$$G_i^{Fils} = \begin{cases} G_i^{Parent1} & si \quad r > p_c \\ G_i^{Parent2} & \text{sinon} \end{cases}$$

r : nombre aléatoire entre 0 et 1



Algorithme génétique – *algorithme*

Etape n°4.2 : Croisement Uniforme

Afin d'explorer de nouvelles zones, il faut générer de nouveaux individus.
A partir de 2 parents, on va générer 2 nouveaux individus en échangeant de l'information entre les parents

$$G_i^{Fils} = \begin{cases} G_i^{Parent1} & si \quad r > p_c \\ G_i^{Parent2} & \text{sinon} \end{cases}$$

r : nombre aléatoire entre 0 et 1

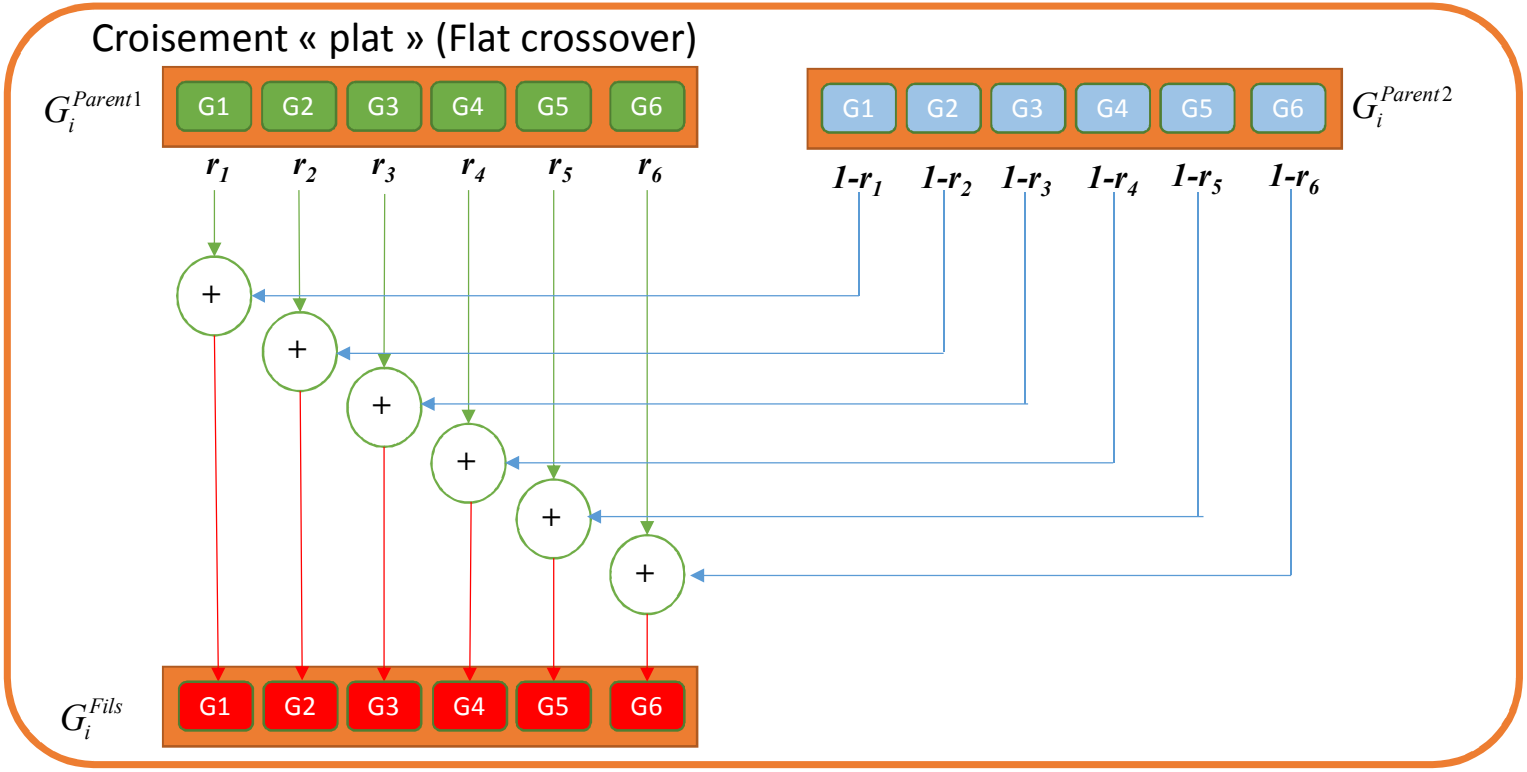
```
function Fils=CroisementUniforme(Parents,nfils,proba)
% Attends 2xnfils parents
nGenome=size(Parents,1);
Fils=zeros(nGenome,nfils);
for i=1:nfils
    % Numéros des parents
    i1=2*(i-1)+1;
    i2=2*i;
    for j=1:nGenome
        if rand>proba
            Fils(j,i)=Parents(j,i1);
        else
            Fils(j,i)=Parents(j,i2);
        end
    end
end
end
```

Algorithme génétique – *algorithme*

Etape n°4.2 : Croisement plat

Afin d'explorer de nouvelles zones, il faut générer de nouveaux individus.
A partir de 2 parents, on va générer 2 nouveaux individus en échangeant de l'information entre les parents

$$G_i^{Fils} = G_i^{Parent1} \cdot r + (1-r) G_i^{Parent2}$$ r : nombre aléatoire entre 0 et 1



Algorithme génétique – *algorithme*

Etape n°4.2 : Croisement plat

Afin d'explorer de nouvelles zones, il faut générer de nouveaux individus.
A partir de 2 parents, on va générer 2 nouveaux individus en échangeant de l'information entre les parents

$$G_i^{Fils} = G_i^{Parent1} \cdot r + (1-r) G_i^{Parent2}$$ r : nombre aléatoire entre 0 et 1

```
function Fils=CroisementPlat(Parents,nfils)
% Attends un nombre de parents égal à 2 x nfils
nGenome=size(Parents,1);
Fils=zeros(nGenome,nfils);
for i=1:nfils
    % Numéros des parents
    i1=2*(i-1)+1;
    i2=2*i;
    % Croisement
    r=rand(nGenome,1);
    Fils(:,i)=r*Parents(:,i1)+(1-r)*Parents(:,i2);
end
```

Algorithme génétique – *algorithme*

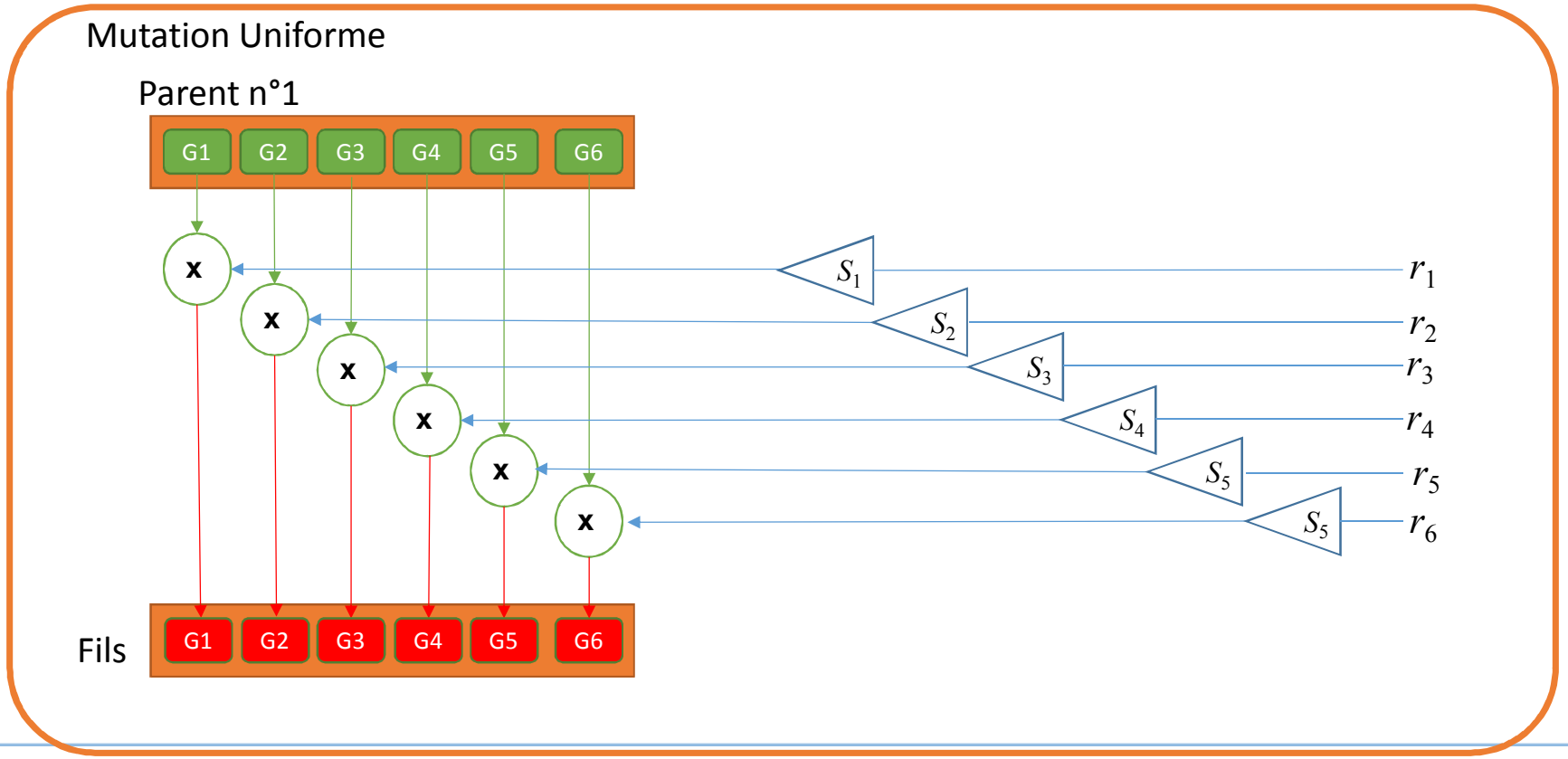
Etape n°4.2 : Mutations

L'objectif est d'altérer un parent pour générer un fils. $G_i^{Fils} = G_i^{Parent} + r \cdot S_i$

r : nombre aléatoire $\in \mathbb{R}$

S_i : Echelle du gène n°i

Il peut être utile de diminuer progressivement le facteur d'échelle à chaque itération *k*: $S_i^{k+1} = S_i^k \alpha$



Algorithme génétique – *algorithme*

Etape n°4.2 : Mutations

L'objectif est d'altérer un parent pour générer un fils. $G_i^{Fils} = G_i^{Parent} + r \cdot S_i$

r : nombre aléatoire $\in \mathbb{R}$

S_i : Echelle du gène n°i

Il peut être utile de diminuer progressivement le facteur d'échelle à chaque itération *k*: $S_i^{k+1} = S_i^k \alpha$

```
function Fils=CroisementUniforme(Parents,nfils,proba)
% Attends 2xnfils parents
nGenome=size(Parents,1);
Fils=zeros(nGenome,nfils);
for i=1:nfils
    % Numéros des parents
    i1=2*(i-1)+1;
    i2=2*i;
    for j=1:nGenome
        if rand>proba
            Fils(j,i)=Parents(j,i1);
        else
            Fils(j,i)=Parents(j,i2);
        end
    end
end
end
```

Algorithme génétique – *algorithme*

Algorithme

```
clear all;
close all;
clc;

f=@(x) 20+ x(1,:).^2+x(2,:).^2  - 10* (cos(2*pi*x(1,:))+cos(2*pi*x(2,:)));

nindividus=20;
nGenes=2;
Scale=3;
nElites=3;
alpha=0.9;
ProbabiliteCroisement=0.7;

nCrossovers=ceil((nindividus-nElites)*0.6);
nMutations=nindividus-nElites-nCrossovers;
nParents=nCrossovers*2+nMutations;


Population=GenerePopulationInitiale(f,nGenes,nindividus,Scale);
```

Fonction à minimiser

Paramètres de l'algorithme

Algorithme génétique – *algorithme*

Programme principal

```
iter=0;
NiterMax=100;
while iter<NiterMax
    Fitness=f(Population);
    Perf=EvaluationPerf(Fitness);

    % Selection des élites
    Elites=SelectionElites(Population,Fitness,nElites);

    % Sélection des parents
    Parents=SelectionParents(Perf,Population,nParents);

    %FilsCroisements=CroisementUniforme(Parents(:,1:nCrossovers*2),nCrossovers,ProbabiliteCroisement);
    FilsCroisements=CroisementPlat(Parents(:,1:nCrossovers*2),nCrossovers);

    Scale=Scale*alpha;

    % Construction de la nouvelle population
    Population=[Elites FilsCroisements FilsMutations];

    % Affichage
    Affichage(iter,Population,Fitness,f);

    % Iteration
    iter=iter+1;
end
```

```
[Best,iMin]=min(Fitness);
x=Population(:,iMin);
fprintf('Meilleur individu trouvé : %.2g\n',Best);
```

Garde le meilleur individu

Algorithme génétique – *algorithme*

Algorithme

```
function Affichage(iter,Population,Fitness,f)
figure(10);
if iter==1
    subplot(1,2,1);
    xlabel('iteration');
    ylabel('Fitness');
    hold on;

    subplot(1,2,2);
    X=linspace(-5,5,100);
    Y=linspace(-5,5,101);
    Z=zeros(length(X),length(Y));
    for i=1:length(X)
        for j=1:length(Y)
            Z(i,j)=f([X(i); Y(j)]);
        end
    end
    surf(X,Y,Z','EdgeColor','none');
    %contourf(X,Y,Z');
    grid on;xlabel('x');ylabel('y');

    hold on;
    xlabel('x1');
    ylabel('x2');
    h=plot3(Population(1,:),Population(2,:),Fitness,'r*');
    set(h,'tag','dataFitness');
```

Affecte « Tag » (nom) aux données

```
else
    subplot(1,2,1);
    h=findobj('tag','dataFitness');
    set(h,'XData',Population(1,:),'YData',Population(2,:),'ZData',Fitness);
end
subplot(1,2,1);
plot(iter,Fitness,'r*');
hold on;
drawnow;
pause(0.5)
end
```

Retrouve l'objet par son « Tag » (nom)

Modifie le graphique sans retracer toute la figure